28-战略设计：如何划分系统的模块？

你好，我是郑晔！

上一讲，我们已经初步认识了 DDD ，知道了支撑 DDD 最核心的就是通用语言和模型驱动设计的方法。我们在上一讲已经讲了建立通用语言的方法，接下来，就该进行模型的设计了。

在模型设计中，DDD 又分成了两个阶段，战略设计和战术设计。这一讲，我们先来聊聊战略设计，下一讲再来谈战术设计。

战略设计，这个名字听上去有点高大上。而且，战略设计包含很多的概念，比如，子域、限界上下文和上下文映射图等等。这让很多人有些望而却步。虽然概念看似很多，但只要有一条主线将它们贯穿起来，这些概念也不难理解。

我们可以先把这些概念做一个划分，分为**做业务的划分和落地成解决方案**两个部分，也就是说，战略设计中的概念，一部分是为了将不同的业务区分开来，也就是要将识别出来的业务概念做一个划分，另一部分则是将划分出来的业务落实到真实的解决方案中。

好，我们接下来就先来看看战略设计中的这些概念到底是怎么回事。

业务概念的划分

我们前面说过，软件开发就是在解决问题，所以，一方面，我们要知道要解决的问题是什么；另一方面，我们要知道怎么去解决问题。

我们要解决的问题就是领域问题，在 DDD 中，有几个概念是与领域相关的，比如，子域、核心域、支撑域、通用域等。其实，它们说的都是一件事，就是如何先把问题从大面上进行分解。

领域驱动设计这个名字里面，排在第一位的是**领域（Domain）**，它就对应着要解决的问题。正如我们一直说的，软件开发是解决问题，而解决问题要分而治之。所谓分而治之，就是要把问题分解了，对应到领域驱动设计中，就是要把一个大领域分解成若干的小领域，而这个分解出来的小领域就是**子域（Subdomain）**。

我们在上一讲中说，领域驱动设计首先要建立起一套通用语言，这样一来，我们就拥有了各种各样的词汇，它们对应着模型。接下来，我们就要给这些词汇做个分类，而分类就是要把它们划分到不同的子域中去。这里面的关键就在于，要找出不同的关注点。没错，还是分离关注点。

比如，我要做一个项目管理软件，就需要有用户、有项目、有团队，不同的人还要扮演不同的角色。第一步，我们至少可以先把身份管理和项目管理这两件事分开，因为它们的关注点是不同的。身份管理关注的是用户的身份信息，诸如用户名密码之类的，而项目管理关注的重点是项目和团队之类的。所以，我们这里有了两个子域：身份管理和项目管理。

如果直接给你看结果，你可能会觉得很好理解。但是，划分出不同的子域还是比较容易出问题的，因为有一些概念并不容易区分。比如，用户应该怎么划分呢？放在身份管理是合适的，但项目管理也要用到用户。

幸好，我们已经学习了单一职责原则，它给了我们一个重要的思考维度，变化从何而来。不同角色的人会关注不同的变化，所以，我们知道虽然我们用的词都是“用户”，但我们想表达的含义却是不同的，我们最好

将这些不同的含义分开，也就是将不同的角色分开。

比如，在身份管理中，它是“用户”，而在项目管理中，它就成了“项目成员”。所以，我们划分子域实际上就是在把不同的概念区分开来，让它们各归其位。

对于一个真实项目而言，划分出来的子域可能会有很多，但并非每个子域都一样重要。所以，我们还要把划分出来的子域再做一下区分，分成核心域（Core Domain）、支撑域（Supporting Subdomain）和通用域（Generic Subdomain）。

核心域是整个系统最重要的部分，是整个业务得以成功的关键。关于核心域，Eric Evans 曾提出过几个问题，帮我们识别核心域：

- 为什么这个系统值得写？
- 为什么不直接买一个？
- 为什么不外包？

如果你对这几个问题的回答能够帮你找到这个系统非写不可的理由，那它就是你的核心域。

什么是支撑域呢？有一些子域不是你的核心竞争力，但却是系统不得不做的东西，市场上也找不到一个现成的方案，这种子域就是支撑域。比如，我们要做一个排行榜功能，可能根据各种信息做排名，这种东西没有人会按照你的需要做出一个，对你来说，又是扩展自己系统的重要一步，它就是一个支撑域。

还有一种子域叫通用域，就是行业里通常都是这么做，即便不自己做，也并不影响你的业务运行。比如，很多 App 要给用户发通知，这样的功能完全可以买一个服务来做，丝毫不影响你的业务运行。它就是一个通用域。

我们之所以要区分不同的子域，关键的原因就在于，我们可以决定不同的投资策略。核心域要全力投入，支撑域次之，通用域甚至可以花钱买服务。

业务概念的落地

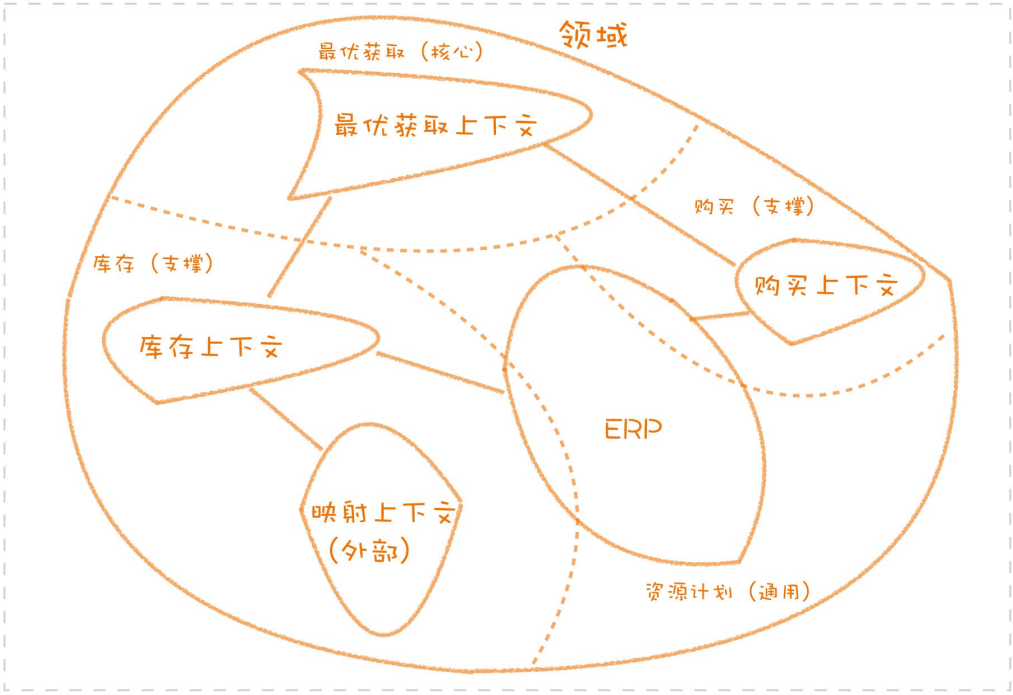
通过划分子域，区分核心域、支撑域和通用域，我们把DDD在问题层面的概念已经说清楚了。接下来，就要进入到解决方案层面了。

我们现在有了切分出来的子域，怎样去落实到代码上呢？首先要解决的就是这些子域如何组织的问题，是写一个程序把所有子域都放在里面呢，还是每个子域做一个独立的应用，抑或是有一些在一起，有一些分开。

这就引出了领域驱动设计中的一个重要的概念，限界上下文（Bounded Context）。

限界上下文，顾名思义，它形成了一个边界，一个限定了通用语言自由使用的边界，一旦出界，含义便无法保证。比如，同样是说“订单”，如果不加限制，你很难区分它是用在哪种场景之下。而一旦定义了限界上下文，那交易上下文的“订单”和物流上下文的“订单”肯定是不同的。原因就在于，订单这个说法，在不同的边界内，含义是不一样的。

注意，子域和限界上下文不一定是一一对应的，可能在一个限界上下文中包含了多个子域，也可能在一个子域横跨了多个限界上下文。



前面我们说了限界上下文是在解决方案层面的，所以，很自然地，我们就可以把限界上下文看作是一个独立的系统。很多团队做微服务的时候，最纠结的问题就是如何划分服务边界，而限界上下文的出现刚好与微服务的理念契合，每个限界上下文都可以成为一个独立的服务。

限界上下文的重点在于，它是完全独立的，不会为了完成一个业务需求要跑到其他服务中去做很多事，而这恰恰是很多微服务出问题的点，比如，一个业务功能要调用很多其他系统的功能。

有了对限界上下文的理解，我们就可以把整个业务分解到不同的限界上下文中，但是，尽管我们拆分了系统，它们终究还是一个系统，免不了彼此之间要有交互。

比如，一个用户下了订单，这是在订单上下文中完成的。那接下来，用户要去支付，这是在支付上下文中完成的。我们肯定要通过某种途径让订单上下文的一些信息发送到支付上下文里的。

所以，我们就要有一种描述方式，将不同限界上下文之间交互的方式描述出来，这就是上下文映射图（Context Map）。DDD 给我们提供了一些描述这种交互的方式，比如：

- 合作关系（Partnership）；
- 共享内核（Shared Kernel）；
- 客户-供应商（Customer-Supplier）；
- 跟随者（Conformist）；
- 防腐层（Anticorruption Layer）；
- 开放主机服务（Open Host Service）；
- 发布语言（Published Language）；
- 各行其道（Separate Ways）；
- 大泥球（Big Ball of Mud）。

之所以有这么多不同的交互方式，主要是为了让你在头脑中仔细辨认一下，看看限界上下文之间到底在以怎

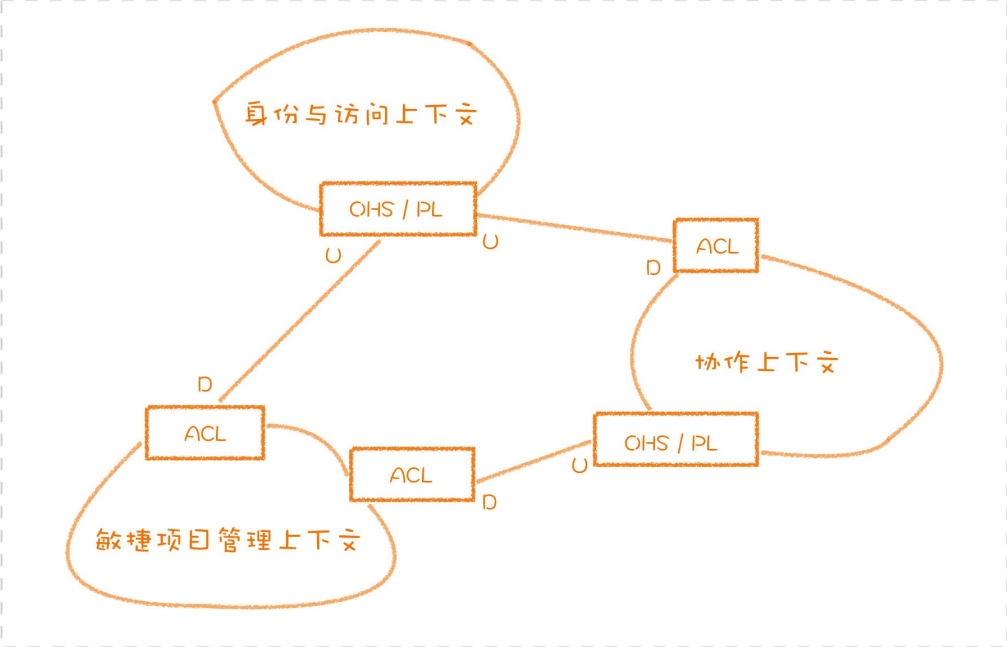
样的方式进行交互。

当然这么多交互方式，想一次性记住也是不现实的，有些甚至是你要规避的，比如大泥球。如果说这么多交互方式你必须要记住一个的话，那就是防腐层（Anticorruption Layer）。

防腐层是最具防御性的一种关系，简言之，就是指我们要在外部模型和内部模型之间建立起一个翻译层，将外部模型转化为内部模型。我在第1讲给你讲了一个因为没有建立防腐造成的问题。所以，但凡有可能，就要建立防腐层，将外部模型完全隔离开。

当我们知道了不同的限界上下文之间采用哪种交互方式之后，不同的交互方式就可以落地为不同的协议。现在最常用的几种协议有REST API、RPC 或是消息队列，我们可以根据实际情况进行选择。

在我们定义好不同的限界上下文，将它们之间的交互呈现出来之后，我们就得到了一张**上下文映射图（Context Map）**。上下文映射图是可以，而这往往是很多团队欠缺的。



总结时刻

今天，我们主要讲了DDD 中的战略设计。战略设计中的概念主要是为了**做业务的划分和落地成解决方案**。

首先业务的划分，我们要把识别出来的模型做一个分类，把它们放置到不同的子域中。划分子域的出发点就是不同的关注点，也就是不同的变化来源。

划分出来的子域有着不同的重要程度，我们将它们再分为核心域、支撑域和通用域。做出这种区分，主要是为了针对它们各自的特点，决定不同的投入。

有了不同的领域划分，我们还要把这些领域映射到解决方案上，这就引出了限界上下文。限界上下文限定了模型的使用边界，它可以成为一个独立的系统。如果对应到微服务中，每一个限界上下文可以对应成一个微服务。

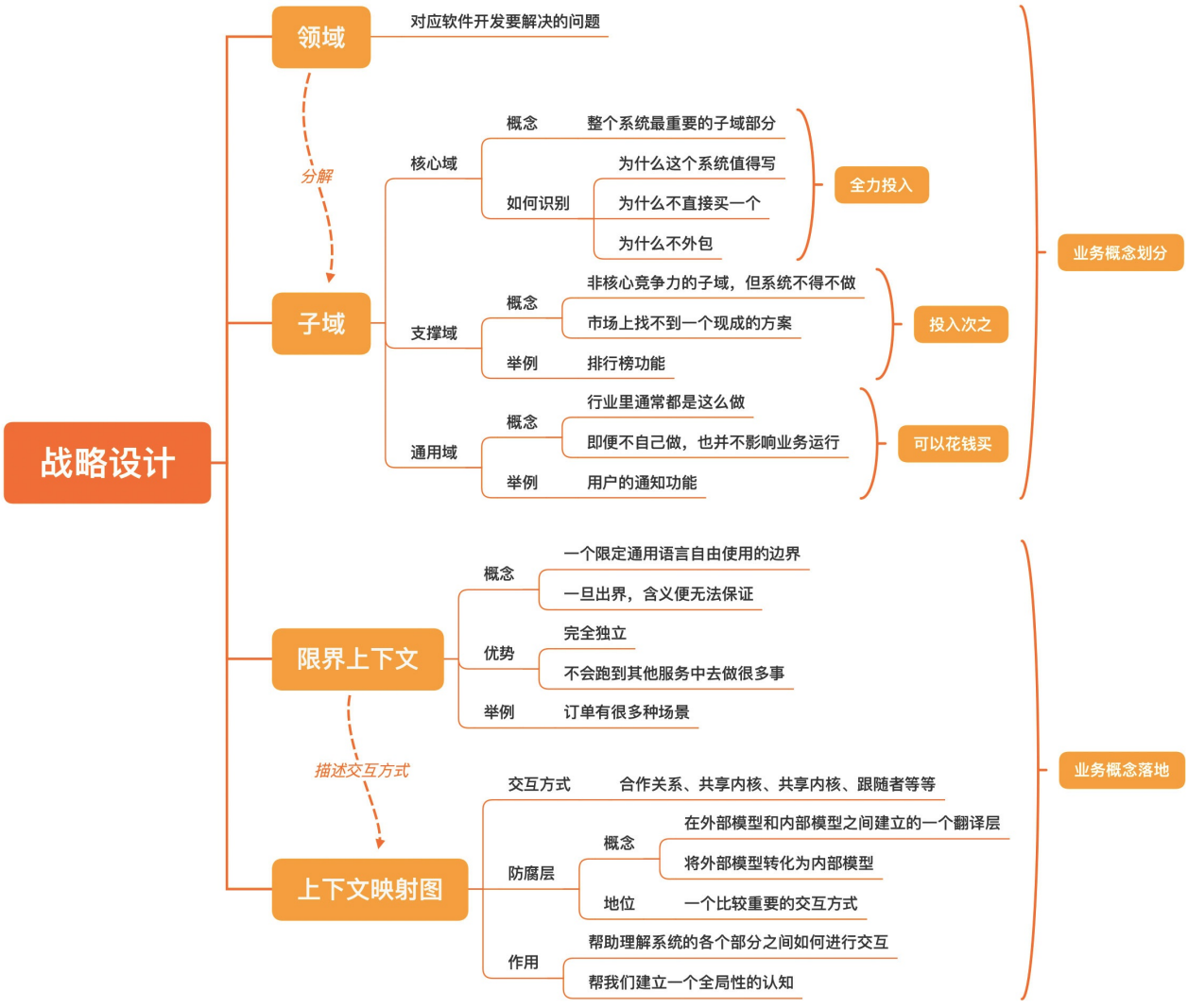
上下文映射图定义了不同上下文之间的交互方式，如果你只能记住一种交互方式的话，就应该记住防腐层。

按照我们之前介绍的了解软件设计的思路，建立起通用语言之后，我们就找到了主要的模型，通过战略设计，我们可以把识别出来的模型放到不同的限界上下文中，就相当于把模型做了分组。然后，我们需要定义出一些接口，让不同的模型之间可以交互，我们也就有了一张上下文映射图。

这样一来，我们就把之前学习的知识和新的知识建立起了连接。

我们有了模型，有了接口，接下来就该深入到实现中。下一讲，我们就要进一步了解 DDD 的实现：战术设计。

如果今天的内容你只能记住一件事，那请记住：**战略设计，就是将不同的模型进行分组**。



极客时间

思考题

最后，我想请你分享一下，你的项目在模型的分组上哪些地方做得好，哪些地方做得不够好呢？欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- 阳仔 2020-07-31 09:47:15
  总结一下
  战略设计首先要把业务概念做一个划分，
  这又分为业务划分和解决方案落地
  业务划分将领域问题拆分为多个子领域问题，根据重要程度不一样又分为核心域、支撑域、通用域。
  解决方案落地则需要做到把拆分的各个子领域问题对应到解决方案中，限定上下文就这样的一个桥梁，而上下文映射则定义了限定上下文之间的交互方式。
  最重要的交互方式就是建立防腐层，它的作用是把外部模型和内部模型之间的交互进行转化（这个感觉有点类似适配器模式的意思）